🔄 Detailed Flow Explanation for Each Test Case

🔐 Authentication Flow

Test Case 1: CNIC Verification Flow

User Input: `"hi"` → `"2"` → `"42502-5550916-6"`

Step-by-Step Flow:

1. **"hi" - Initial Greeting**

```
Facebook → Webhook → process_user_message_api_based()
├── verification_stage = "not_verified"
├── is_greeting_message("hi") = True
├── API Call: POST /detect_initial_choice
│   └── enhanced_ai_agent.detect_initial_choice()
│   └── Returns: choice_detected="none", shows menu
└── Response: "Welcome! 1. Bank info 2. Account access"
```

2. **"2" - Account Access Choice**

```
Facebook → Webhook → process_user_message_api_based()
├── verification_stage = "not_verified"
├── is_greeting_message("2") = False
├── API Call: POST /detect_initial_choice
│   └── enhanced_ai_agent.detect_initial_choice()
│   └── Returns: choice_detected="2"
└── Response: "Please provide your CNIC in format 12345-1234567-1"
```

3. **"42502-5550916-6" - CNIC Verification**

```
Facebook → Webhook → handle_cnic_verification_api_based()
├── Extract CNIC: api_call_extract_cnic() OR direct regex match
├── API Call: POST /verify_cnic
│   └── MongoDB Query: transactions.find_one({"cnic": "42502-5550916-6"})
│   └── Find accounts: transactions.distinct("account_number", {"cnic": cnic})
│   └── Returns: {status: "success", user: {name, cnic, accounts}}
├── Update state: set_user_verification_stage(sender_id, "CNIC_VERIFIED")
├── API Call: POST /handle_cnic_verification_success
└── Response: "Welcome Ali! Please select your account: 001-0156-000654321 (PKR), 001-0156-000654789
(USD)"
```

Test Case 2: Account Selection Flow

User Input: "PKR account"

Step-by-Step Flow:

```
Facebook → Webhook → handle_account_selection_api_based()
├── verification_stage = "cnic_verified"
├── API Call: POST /enhanced_account_selection
│   └── enhanced_ai_agent.enhanced_account_selection_v2()
│   └── Currency detection: "PKR" → matches account ending in 4321
│   └── Returns: {matched_account: "001-0156-000654321", selection_method: "currency"}
├── API Call: POST /select_account
│   └── MongoDB Query: transactions.find_one({"cnic": cnic, "account_number": account})
│   └── Verify account belongs to user
├── Update state: set_user_verification_stage(sender_id, "ACCOUNT_SELECTED")
├── API Call: POST /handle_account_confirmation
└── Response: "Perfect! Account ***-***-4321 is ready, Ali. How can I help?"
```

💰 Balance and Transaction Queries

Test Case 3: Balance Query Flow

User Input: "what's my balance?"

Step-by-Step Flow:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── API Call: POST /process_query
│   └── NOT bank info query (doesn't match keywords)
│   └── enhanced_ai_agent.process_query()
│       ├── Intent Detection: "account_query"
│       ├── Mode Setting: user_modes[account] = "account"
│       ├── Query Classification: "simple" (basic balance check)
│       └── Route to: simple_balance_query()
│           └── MongoDB Query: transactions.find_one(
│               {"account_number": account},
│               sort=[("date", -1), ("_id", -1)]
│           )
│           └── Extract: account_balance, account_currency
│           └── LLM Response Generation:
│               └── Type: "balance_info"
│               └── Data: {balance: 75000, currency: "PKR", account: "***-4321"}
└── Response: "Your current balance is PKR 75,000 in account ***-***-4321, Ali."
```

Test Case 5: Simple Transaction List Flow

User Input: "show my transactions"

Step-by-Step Flow:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── API Call: POST /process_query
│   └── enhanced_ai_agent.process_query()
│       ├── Intent Detection: "account_query"
│       ├── Query Classification: "simple"
│       │   └── No date filters, no complex analysis needed
│       └── Route to: get_simple_transactions()
│           ├── MongoDB Query: transactions.find(
│           │   {"account_number": account}
│           │   ).sort([("date", -1)]).limit(10)
│           ├── Format transactions for display
│           └── LLM Response Generation:
│               └── Type: "simple_transactions"
│               └── Format: "1. Date | Description | Amount"
│               └── Limit: Under 1800 characters for Facebook
└── Response: "Here are your recent transactions:\n1. Jul 15 | Grocery Store | Debit 2500 PKR\n2. Jul 14 | Salary | Credit 45000 PKR\n..."
```

🔍 Complex Analysis Flows

Test Case 8: Highest Transaction Flow

User Input: "what's my highest transaction?"

Step-by-Step Flow:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── API Call: POST /process_query
│   └── enhanced_ai_agent.process_query()
│       ├── Intent Detection: "account_query"
│       ├── Query Classification: "analysis"
│       │   └── Keywords: "highest", "max" detected
│       │   └── requires_pipeline = True
│       └── Route to: process_transactions_with_context()
│           ├── LLM Pipeline Generation:
│           │   └── Prompt: "Generate MongoDB pipeline for highest transaction"
│           │   └── Generated Pipeline: [
│           │       {"$match": {"account_number": account}},
│           │       {"$sort": {"transaction_amount": -1}},
│           │       {"$limit": 1}
│           │   ]
│           ├── API Call: POST /execute_pipeline
│           │   └── MongoDB: transactions.aggregate(pipeline)
│           │   └── Returns: [{transaction with highest amount}]
│           ├── Store Context: transaction_contexts[account] = results
│           └── LLM Response Generation:
│               └── Type: "analysis_transactions"
│               └── Data: {transactions: [...], analysis_result: "highest"}
└── Response: "Your highest transaction was PKR 45,000 - Salary credit on July 14th, Ali."
```

Test Case 9: Monthly Comparison Flow

User Input: "compare my spending in May vs April"

Step-by-Step Flow:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
 ├── verification_stage = "account_selected"
 ├── API Call: POST /process_query
 │   └── enhanced_ai_agent.process_query()
 │       ├── Intent Detection: "account_query"
 │       ├── Query Classification: "complex"
 │       │   └── Keywords: "compare", "vs", months detected
 │       │   └── requires_pipeline = True
 │       └── Route to: process_transactions_with_context()
 │           ├── LLM Pipeline Generation:
 │           │   └── Prompt: "Generate pipeline for comparing May vs April spending"
 │           │   └── Generated Pipeline: [
 │           │     {"$match": {"account_number": account, "type": "debit"}},
 │           │     {"$addFields": {"month": {"$month": "$date"}}},
 │           │     {"$match": {"month": {"$in": [4, 5]}}},
 │           │     {"$group": {
 │           │        "_id": "$month",
 │           │        "total_spending": {"$sum": "$transaction_amount"},
 │           │        "transaction_count": {"$sum": 1}
 │           │     }},
 │           │     {"$sort": {"_id": 1}}
 │           │   ]
 │           ├── API Call: POST /execute_pipeline
 │           │   └── MongoDB: transactions.aggregate(pipeline)
 │           │   └── Returns: [
 │           │     {_id: 4, total_spending: 25000, transaction_count: 15},
 │           │     {_id: 5, total_spending: 32000, transaction_count: 18}
 │           │   ]
 │           ├── Store Context: transaction_contexts[account] = results
 │           └── LLM Response Generation:
 │               └── Type: "complex_analysis"
 │               └── Data: {comparison: months, analysis_type: "spending"}
 │               └── Analysis: Calculate differences, percentages
 └── Response: "Spending Comparison:\n\nApril: PKR 25,000 (15 transactions)\nMay: PKR 32,000 (18
transactions)\n\nYou spent PKR 7,000 more in May (28% increase), Ali."
```

🔄 Contextual Memory Flow

Test Case 14: Follow-up Query Flow

First Query: "show my May transactions" **Follow-up:** "which one was the highest?"

Step-by-Step Flow:

First Query:

1. Enhanced AI processes "show my May transactions"
2. Generates and executes pipeline for May transactions
3. Stores results in: transaction_contexts[account] = {query: "May transactions", results: [...]}
4. Responds with May transaction list

Follow-up Query:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── API Call: POST /process_query
│   └── enhanced_ai_agent.process_query()
│       ├── Intent Detection: "account_query"
│       ├── Query Classification: "contextual"
│       │   └── Keywords: "which one", "highest" + context exists
│       │   └── Context Reference: transaction_contexts[account] exists
│       └── Route to: process_transactions_with_context()
│           ├── Load Previous Context: transaction_contexts[account]
│           ├── Context Analysis: "User asking about highest from May transactions"
│           ├── Process on Stored Results (not new DB query):
│           │   └── Find highest transaction from stored May results
│           │   └── No new MongoDB query needed
│           └── LLM Response Generation:
│               └── Type: "contextual_analysis"
│               └── Reference: "From your May transactions"
│               └── Data: {highest_transaction: {...}, context: "May results"}
└── Response: "From your May transactions, the highest was PKR 8,500 for Shopping Mall on May 22nd."
```

💸 Money Transfer Flow

Test Case 12: Money Transfer Flow

User Input: "transfer 500 PKR to John"

Step-by-Step Flow:

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── API Call: POST /process_query
│   └── enhanced_ai_agent.process_query()
│       ├── Intent Detection: "transfer_money"
│       │   └── Keywords: "transfer", amount, recipient detected
│       ├── Extract Transfer Details:
│       │   └── Amount: 500, Currency: PKR, Recipient: "John"
│       ├── Store Transfer State: transfer_states[account] = {
│       │   amount: 500, currency: "PKR", recipient: "John",
│       │   status: "pending_confirmation"
│       │   }
│       └── Generate Confirmation Request:
│           └── "Confirm transfer of PKR 500 to John? Reply 'yes' to confirm."
```

Follow-up: "yes"

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── verification_stage = "account_selected"
├── Load Transfer State: transfer_states[account]
├── API Call: POST /transfer_money
│   ├── Validation Checks:
│   │   ├── Amount > 0? ✓
│   │   ├── Currency matches account? ✓
│   │   ├── Sufficient balance?
│   │   │   └── MongoDB: Get latest transaction for current balance
│   │   │   └── Check: current_balance >= transfer_amount
│   │   └── All validations pass ✓
│   ├── Create Transfer Transaction:
│   │   └── MongoDB: transactions.insert_one({
│   │       name, cnic, account_number, date: now(),
│   │       type: "debit", description: "Transfer to John",
│   │       amount_deducted: 500, new_balance: old_balance - 500
│   │   })
│   └── Return: {status: "success", new_balance, transaction_id}
├── Clear Transfer State: del transfer_states[account]
└── Response: "✅ Transfer successful! PKR 500 sent to John. New balance: PKR 74,500. Transaction ID: 64f..."
```

🚨 Error Handling Flows

Invalid CNIC Flow

User Input: "invalid-cnic-123"

```
Facebook → Webhook → handle_cnic_verification_api_based()
├── Extract CNIC: api_call_extract_cnic("invalid-cnic-123")
│   └── Returns: None (no valid CNIC found)
├── Regex Check: r'^\d{5}-\d{7}-\d$' → No match
├── API Call: POST /handle_invalid_cnic_format
│   └── enhanced_ai_agent.response_generator.generate_response()
│   └── Type: "cnic_validation_error"
└── Response: "Please provide your CNIC in the correct format: 12345-1234567-1. For example: 42502-5550916-6"
```

Insufficient Balance Flow

User Input: "transfer 100000 PKR to Sarah" (amount > balance)

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── Transfer processing initiated
├── API Call: POST /transfer_money
│   ├── Get Current Balance: 75000 PKR
│   ├── Check: 100000 > 75000 → Insufficient funds
│   └── Return: {status: "fail", reason: "Insufficient PKR balance..."}
└── Response: "❌ Insufficient PKR balance. Available: 75,000.00, Required: 100,000.00. Please try a smaller amount."
```

📱 Message Splitting Flow

Long Response Flow

User Input: "show all my transactions from March to May with analysis"

```
Facebook → Webhook → handle_authenticated_queries_api_based()
├── Process complex query → Generates 2500 character response
├── _ensure_message_length() detects: len(response) > 1900
├── send_message() → _send_long_message_in_parts()
│   ├── Split Strategy:
│   │   ├── Try splitting by "\n\n" (paragraphs) first
│   │   ├── If section still too long → split by lines
│   │   └── If line too long → split by words
│   ├── Part 1 (1800 chars): "Here are your transactions from March to May:\n\n**March Transactions:**\n1. Mar 1 | Salary | Credit 45000 PKR\n..."
│   ├── Send Part 1 → Facebook API
│   ├── time.sleep(0.3) → Brief delay
│   ├── Part 2 (700 chars): "**Analysis Summary:**\nTotal spent: PKR 67,500\nHighest: PKR 8,500 (Shopping)\nAverage: PKR 1,250 per transaction"
│   └── Send Part 2 → Facebook API
└── User receives 2 separate messages in sequence
```

🔄 Session Management Flow

Exit Command Flow

User Input: `"exit"`

```
Facebook → Webhook → process_user_message_api_based()
├── Check: user_message.strip().lower() == "exit" → True
├── Get User Info: get_user_account_info(sender_id)
├── API Call: POST /handle_session_end
│   └── enhanced_ai_agent.response_generator.generate_response()
│   └── Type: "session_end"
├── Clear Session: clear_user_state(sender_id)
│   ├── Remove from authenticated_users
│   ├── Clear verification stage
│   └── Reset to initial state
└── Response: "Thank you for using Best Bank, Ali. Your session has ended securely. Have a great day! 👋"
```

⚡ Performance Characteristics

Response Time Flows

1. **Simple Balance Query:** ~100-500ms

```
Webhook (50ms) → API Call (10ms) → DB Query (30ms) → LLM Response (200ms) → Facebook (50ms)
```

2. **Complex Analysis:** ~2-5 seconds

   Webhook (50ms) → API (10ms) → LLM Pipeline Gen (2000ms) → DB Aggregate (500ms) → LLM Response (1000ms) → Facebook (50ms)

3. **Contextual Follow-up:** ~1-2 seconds

   Webhook (50ms) → API (10ms) → Context Load (10ms) → Analysis (800ms) → LLM Response (500ms) → Facebook (50ms)

🎯 Key Takeaways

- **Simple queries** use direct database access (fast)
- **Complex queries** generate MongoDB pipelines via LLM (slower)
- **Contextual queries** reuse stored results (medium speed)
- **All communication** goes through HTTP API calls (pure microservices)
- **Message splitting** handles Facebook's character limits automatically
- **State management** tracks user progress through authentication flow
- **Error handling** provides natural, helpful error messages at each step

Each flow maintains **conversation context** and **natural language understanding** while ensuring **secure** and **reliable** banking operations.